Experiences with applying a genetic algorithm to determine an information systems architecture

Anwendung eines Genetischen Algorithmus zur Bestimmung der Architektur von Informationssystemen

G. F. Knolmayer, J.-P. Gerber

Institut für Wirtschaftsinformatik, Abteilung Information Engineering, Universität Bern, Engehaldenstrasse 8, CH-3012 Bern, Switzerland
(Fax: ++41/31/631 46 82, e-mail: knolmayer@ie.iwi.unibe.ch)

Abstract. While determining information systems architectures (ISA), business systems planning (BSP) is a well-known method to join processes and data classes to subsystems. BSP matrices have generally been rearranged without describing the underlying methods. Meanwhile, various techniques have been developed for solving the ISA problem. Since exact optimization methods often fail to provide results for large ISA problems, different heuristics have been applied. A new heuristic for solving the ISA problem is the application of genetic algorithms (GA). This paper examines the application of a simple GA to the ISA problem and compares the results of applying the GA with those obtained by exact methods.

Zusammenfassung. Zur Entwicklung von Architekturen von Informationssystemen (ISA) wird vielfach das Business-Systems-Planning-(BSP)-Konzept vorgeschlagen. Ein Teilproblem dieses Planungskonzepts besteht darin, unter Berücksichtigung von Optimalitätskriterien Unternehmensprozesse und Datenbestände zu möglichst voneinander unabhängigen Teilsystemen zusammenzufassen. Da die Leistungsgrenzen von exakten Optimierungsverfahren für dieses Problem rasch erreicht werden, interessiert der Einsatz von heuristischen Verfahren. Zunächst werden das BSP-Problem und die Vorgehensweise genetischer Algorithmen kurz erläutert. Danach wird die Anwendung eines einfachen genetischen Algorithmus auf das BSP-Problem beschrieben. Ein Vergleich mit Ergebnissen exakter Verfahren bildet einen weiteren wichtigen Bestandteil der Untersuchung.

Key words: Information systems architecture, business systems planning, cluster analysis, genetic algorithms

Schlüsselwörter: Architektur von Informationssystemen, Business Systems Planning, Cluster-Analyse, Genetische Algorithmen

Correspondence to: G. F. Knolmayer

1. Introduction

Determining information systems architectures (ISA) to support the competitiveness of enterprises has gained much attention in recent years. Determining an ISA also aims at considering the strategic relevance of data as a corporate resource and improving data quality [cf. 15, 23, 26, 29]. A strategic planning approach is of special importance in designing distributed information systems (IS) and is regarded as the number one of among the ten most critical IS management issues [cf. 19].

The best-known method supporting this strategic task is IBM's business system planning (BSP) [cf. e.g. 1, 8, 26]. BSP is a structured approach which assists an organization in planning its short- and long-term IS requirements. The main purpose of BSP is to support the translation of business strategy into IS strategy by analyzing business information needs and structuring them into several distinct subsystems according to the principle of modularity. The BSP framework proposes 13 steps for conducting the study [8]:

- Gaining executive commitment
- Preparing for the study
- Kickoff meeting
- Defining business processes
- Defining data classes
- Analyzing current systems support
- Determining the executive perspective
- Defining findings and conclusions
- *Defining the information architecture*
- Determining architectural priorities
- Reviewing information resource management
- Developing recommendations and an action plan
- Reporting results.

The definition of an information architecture uses an association matrix, hereafter referred to as BSP matrix, to define connections between processes and data classes. Cross reference matrices which show the relationship between various types of objects are in general an important component of ISA and are implemented in CASE tools

Fig. 1a BSP matrix in original form [20]

Processes \ Data Classes	Customer	Order	Vendor	Product	Routings	Bill of Material	Cost	Parts Master	Raw Material Inventory	Final Goods Inventory	Employee	Sales territory	Financial	Planning	Work in Progress	Facilities	Open Requirements	Machine Load
Business Planning						U							U	C				
Organization Analysis														U				
Review and Control													U	U				
Financial Planning										U			U	C	U			
Capital Acquisition														C				
Research			U										U					
Forecasting	U		U										U		U			
Design and Development	U		C		U		C											
Product Specification Maintenance			U	U		C		C										
Purchasing			C				U											
Receiving			U						U									
Inventory Control								C	C						U			
Workflow Layout				U	U											C		
Scheduling			U	U											C	U		U
Capacity Planning			U		U											U	U	C
Material Requirements			U	U		U											C	
Operations				C											U		U	U
Territory Management	C	U		U														
Selling	U	U		U									C					
Sales Administration		U											U					
Order Servicing	U	C		U														
Shipping		U		U						U								
General Accounting	U		U										U	U				
Cost Planning		U	U			C												
Budget Accounting						U					U		U	U	U			
Personnel Planning											C		U					
Recruiting Development											U							
Compensation											U		U					

a

Fig. 1b BSP matrix in rearranged form for a given sequence of processes

Processes \ Data Classes	Planning	Financial	Product	Bill of Material	Parts Master	Vendor	Raw Material Inventory	Final Goods Inventory	Facilities	Work in Progress	Machine Load	Open Requirements	Routings	Customer	Sales territory	Order	Cost	Employee
Business Planning	C	U																U
Organization Analysis	U																	
Review and Control	U	U																
Financial Planning	C	U								U								U
Capital Acquisition		C																
Research			U											U				
Forecasting	U	U												U	U			
Design and Development			C	C	U										U			
Product Specification Maintenance			U	C	C	U												
Purchasing						C										U		
Receiving						U	U											
Inventory Control							C	C		U								
Workflow Layout			U						C				U					
Scheduling			U		U				U	C	U							
Capacity Planning					U					U	C	U	U					
Material Requirements			U		U	U						C						
Operations										U	U	U	C					
Territory Management			U											C		U		
Selling			U											U	C	U		
Sales Administration															U	U		
Order Servicing			U											U		C		
Shipping			U					U								U		
General Accounting	U			U										U				U
Cost Planning				U												U	C	U
Budget Accounting	U	U								U							U	U
Personnel Planning	U																	C
Recruiting Development																		U
Compensation		U																U

b

Fig. 1. a BSP matrix in original form [20]. **b** BSP matrix in rearranged form for a given sequence of processes

like e.g. KEY (formally named ADW and IEW) for strategic IS planning [cf. e.g. 10, 12, 26]. Not-null elements of the BSP matrix contain the symbol C for processes creating and U for those using data classes. While defining the information architecture it is important to determine subsystems which enclose as many of these symbols as possible. This determination is usually preceded by rearranging rows and columns of the BSP matrix. The rearrangement of given matrices intends to identify business areas or subsystems to be allocated to certain servers in a Client/Server architecture without generating too many interfaces. Thus, the goals to be achieved with the procedures described encompass

- Complexity reduction by defining appropriate subsystems
- Identification of different applications areas e.g. for defining development priorities of building work packets for Year2000 conversion
- Top-down framework for IS development with bottom-up implementation
- Determining a blueprint for distributed IS
- Reduction of number of interfaces and telecommunication costs.

Figure 1 gives an example of a BSP matrix in original and in rearranged form. The resulting subsystems and the data flows between them form an ISA.

2. Methods for restructuring BSP matrices

One step of defining an ISA is to determine subsystems where almost all symbols are positioned within the submatrices at the main diagonal of a restructured BSP matrix. All C-entries must reside within those submatrices to guarantee the assignment of a data class to the process creating it. Early publications [8, 18] provide examples of restructuring BSP matrices without describing the underlying procedures. Methods like ISMOD (Information System Model and Architecture Generator) define an affinity index to determine an appropriate sequence of processes for a *given* sequence of data classes [6, 9, 28] and were implemented in special tools which have been used until 1990 in more than 450 studies [10]. Later on, clustering techniques have been applied to this problem [11, 21].

Recently, different exact as well as heuristic approaches have been considered for two-dimensional restructuring of BSP matrices. One possible exact approach is to develop a quadratic assignment model in which each row and each column of a BSP matrix have to be assigned to a subsystem. After linearization of the quadratic model one may apply mixed-integer programming techniques [13, 14]. Branch-and-bound algorithms (BB) considering the smallest set of all feasible solutions have also been adapted to this type of problem [24]. Experiences with these exact methods show that they cannot be applied to large problems owing to limited computational resources (cf. Sect. 4).

Another possible approach to the ISA problem is the application of heuristics which usually provide good so-

lutions within a manageable amount of time but do not necessarily find the optimal solution. As generally applicable heuristics, genetic algorithms (GA) have been applied for cluster analysis [16] and for solving the quadratic assignment problem [2] with promising results.

This paper proposes GA as a further approach for solving the ISA problem. We briefly review the main principles of GA, introduce a problem simplication by preprocessing the cluster identification algorithm [17], describe the bit string representation and the underlying objective function of the ISA problem and finally compare the results of the GA with results provided by applying exact optimization procedures. Thereby we also contribute to overcome the scientifically unsatisfying situation that most GA applications have never been compared with exact optimization methods at least for problems of small and medium complexity.

3. The application of a simple genetic algorithm to the ISA problem

3.1. Principles of genetic algorithms

Genetic algorithms base on mechanisms of natural reproduction with are, according to Darwin, characterized by the adoption of advantageous modifications and the discard of disadvantageous ones through selection. This evolutionary principle is regarded as powerful and efficient in nature, being probably the best compromise between determination and chance, and has therefore been transferred to construct algorithms for solving complex optimization problems [cf. e.g. 4].

The basic principle of GA is to pass on advantageous genetic information from one generation to the next while ignoring inferior information. In nature the genetic information is carried by a chromosome, in GA usually by a bit string. A set of bit strings forms a population where each bit string represents a possible solution to an optimization problem. An objective function evaluates the quality of each bit string with respect to the underlying optimization problem. The probability for selecting the best bit strings for recombination is proportional to their performance. The most important recombination operators are crossover and mutation. The former exchanges randomly selected portions of bits between two bit strings of the subsequent generation. The latter is designed to invert bits by chance and prevents a premature loss of possibly useful solutions. After applying the recombination operators to a bit string population, a new generation evolves which is expected to consist of some superior bit strings. This procedure of evaluation, selection and recombination is continuously repeated until a user defined condition causes the GA to stop.

3.2. Preprocessing the ISA problem

The ISA problem can be simplified by applying the cluster identification algorithm (CIA) [17] to identify the set of submatrices fulfilling the constraint that all C-entries must reside within the created subsystems. The CIA is an iterated algorithm which draws in step 1 a horizontal line through the first row and then draws in step 2 vertical lines through all C-entries in this row. For each C-entry intersected by the vertical line, a new horizontal line is drawn in step 3. Steps 2 and 3 are repeated until no crossed-once C-entries are left. All rows and columns of crossed-twice C-entries eventually form a cluster or a so-called baseblock [cf. 25]. For the next iteration this baseblock is removed and the CIA continues with step 1 until there are no more C-entries left in the BSP matrix. These submatrices are called baseblocks because they represent an initial solution to the original problem with the largest number of feasible subsystems. The subsystems of the solutions either equal a single baseblock or are combined of several baseblocks. Referring to the original BSP matrix of Fig. 1a, we graphically emphasize in Fig. 2a the rows and columns clustered by the third CIA iteration. In Fig. 2b we show the corresponding transformed matrix after completing all CIA iterations. In the following, the result of applying the CIA sets the basis for further improvements of the ISA.

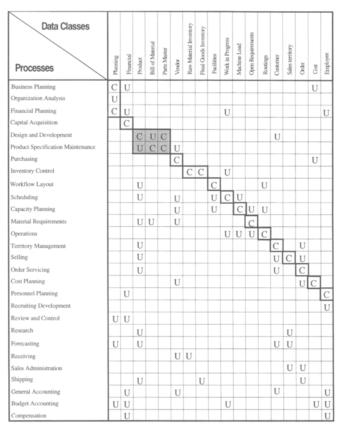### 3.3. Bit string representation of the ISA problem

In the following we discuss the application of a GA to an easier example. Fig. 3a depicts the original BSP matrix and Fig. 3b shows the BSP matrix after applying the CIA with $n = 5$ baseblocks.

To apply a GA to the ISA problem, each feasible combination of baseblocks must be represented by a bit string which can be manipulated by the GA operators. In order to explain the chosen representation, a bit matrix is introduced which contains the bit representation of a possible solution to the ISA problem. The element $m_{ij} \in \{0,1\}$ of the quadratic bit matrix states whether the two baseblocks indexes with i and j are joined ($m_{ij} = 1$) or not ($m_{ij} = 0$). Fig. 4 gives an example of a bit matrix for the above ISA problem. The matrix expresses in binary terms that the baseblocks with the indices 1, 3 and 5 are joined and that the two residual blocks 2 and 4 remain unclustered and form subsystems on their own. Elements at the diagonal of this bit matrix are irrelevant and those beneath it are redundant.

The transformation into the bit string can be easily understood when scanning the matrix elements columnwise. Referring to the matrix in Fig. 4, the corresponding bit string is 0'10'000'1010. The length l of the resulting bit string depends on the number of baseblocks n with $l = n \cdot (n-1)/2$. Hence, with increasing number of baseblocks, the bit string length grows with $O(n^2)$. This may well affect the performance of the GA in terms of time consumption and its abilities to provide good solutions. A further problem that arises in connection with the above representation is that there exist strings which do not represent feasible solutions. Assume that the evolution creates the string 0'10'011'0000 which implies that block 1 is joined to block 3 and blocks 2 and 3 are combined with block 4. Consequently, block 1 has also to be joined with blocks 2 and 4 but the respective bits have not been set. We cope with this inconsistency by applying a 'filling-in'
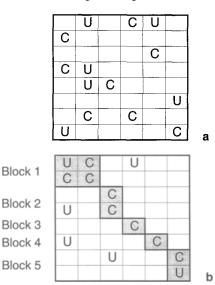
a

Fig. 2. a The third CIA iteration applied to the original BSP matrix. **b** BSP matrix after preprocessing by the CIA

a

b

Fig. 3. a Original BSP matrix. **b** BSP matrix after application of the CIA (initial solution)

	Block 1	Block 2	Block 3	Block 4	Block 5
Block 1		0	1	0	1
Block 2			0	0	0
Block 3				0	1
Block 4					0
Block 5					

Fig. 4. Example of a bit matrix expressing that baseblocks 1, 3 and 5 are combined

strategy that switches all inconsistent bits from 0 to 1; in the above example the bit string 1'11'111'0000 results.

A further drawback of this representation is that the schema theorem [4] might not work properly. It states that the ratio of schemata with short defining length, with low order and with exceptionally good fitness increases exponentially in subsequent generations. This theorem requires an efficient bit string representation which meets two demands. First, the smallest possible alphabet should be used to code the string in order to make it as short as possible. Second, the representation should be chosen such that short and compact schemata can be identified, i.e., parts of a chromosome which represent closely related information should not be dispersed in the bit string. These two recommendations are not satisfied by the above representation of the ISA problem. Nevertheless, this representation was implemented as it is presumably the only meaningful one that allows to use GENESIS 5.0 (Genetic Search Implementation System), a generally available GA program library. GENESIS is task independent and allows to perform all the fundamental GA operations including selection, recombination (mutation and crossover) and evaluation [5].

3.4. Calculating the objective function value

After preprocessing, the ISA problem is reduced to find suitable combinations of baseblocks and positions of the

	Block 1	Block 2	Block 3	Block 4	Block 5
Block 1		- 0.5	0.1	0.1	- 1.8
Block 2			- 0.9	- 0.9	0.1
Block 3				- 0.6	- 0.9
Block 4					- 0.9
Block 5					

Fig. 5. Corresponding delta matrix

resulting subsystems (S) which result in as few interactions as possible between them. Two subsystems interact if there is a U-entry in position k_{is} or k_{rj} of the BSP matrix with $i, j \in S_1$ and $r, s \in S_2$. Thus, the objective function (in terms of GA: fitness) is defined to incorporate as many U-entries as possible within the subsystems. However, the objective function should also prevent that all baseblocks are put together to one or a very small number of big subsystems containing all or too many rows and columns of the original problem as this obviously contradicts the planning intentions. Therefore, we decided to use an objective function that considers this trade-off: joining two baseblocks is evaluated in such a way that each new U-entry in the subsystems is rewarded one point and each new null entry is penalized with p points. The objective is to maximize the fitness, measured by the sum of points:

Objective function value (fitness) $= u - p \cdot b \Rightarrow \max!$ (1)

where u = number of U-entries in the subsystems
     p = penalty value
     b = number of null entries in the subsystems

For our comparisons we fixed the coefficient to $p = 0.3$ because this value provided reasonable compromises in solving some pretest problems. Effects of employing different values of p are described in [25]. We use the parameter value 0.3 also for further explaining the proposed procedure.

A delta matrix, consisting of exactly the same structure as the bit matrix, is introduced for calculating the objective function value. The elements of the delta matrix, also referred to as delta factors, represent the absolute changes of the fitness when two baseblocks are joined. Applying the objective function to the initial solution, the delta factors of all possible combinations of baseblocks can be calculated and stored in this matrix.

Figure 5 gives an example of a delta matrix resulting from the initial solution of the corresponding problem shown in Fig. 3b. If e.g. blocks 1 and 2 are joined, the new subsystem will include five blanks which are penalized with 0.3 points each, and one additional U-entry, scored with 1 point; the resulting delta factor is $d_{12} = 1 - 0.3 \cdot 5 = -0.5$.

The change of fitness as a result of combining baseblocks can be easily computed using the bit matrix and the delta matrix according to formula (2).

Change of fitness $= \sum_i \sum_j d_{ij} \cdot m_{ij}$ (2)

where m_{ij} = element of the bit matrix
    d_{ij} = element of the delta matrix
    i = $1, \ldots, n - 1$
    j = $i + 1, \ldots, n$.

The change of the fitness is the sum of all relevant delta factors. A delta factor is relevant if the two corresponding baseblocks are joined which is indicated by the respective bit set in the bit matrix. The algorithm starts with an initial solution, consisting of n baseblocks, which provides the initial fitness, defined as the sum of all scored blanks and U-entries in the baseblocks:

Initial fitness $= u_b - p \cdot b_b$ (3)

where u_b = total number of U-entries in the baseblocks
    b_b = total number of blank entries in the baseblocks.

To compute an absolute fitness instead of one relative to the initial solution, the initial fitness is added to the accumulated value of all the changes derived from overlapping the delta matrix with the bit matrix:

Fitness $= u_b - p \cdot b_b + \sum_i \sum_j d_{ij} \cdot m_{ij}$.   (4)

Referring to our example, if the GA selects the string 0'00'100'0100, the accumulated change of the fitness is calculated and added to the initial fitness according to (4), resulting in: Fitness $= 2 - 0.3 \cdot 0 + 1 \cdot 0.1 + 1 \cdot 0.1 = 2.2$. Thus, joining the baseblocks 1 and 4 as well as 2 and 5 improves the initial solution.

4. Results of computational tests

The GA and other optimization methods are compared with respect to the quality of the solutions obtained and the consumed computing time for ISA problems of different complexity. The algorithms were run on an IBM RS 60000-320H. Details of the comparisons are given in Table 1.

The GA was run 30 times on each ISA problem. The preliminary number of trials per experiment was intuitively set for each problem and stepwise increased if further improvements of the fitness could be expected. The standard parameter settings of GENESIS (population size = 100; crossover rate = 0.6; mutation rate = 0.001; generation gap = 1.0) were used.

In order to compare the GA with exact optimization algorithms, a specialized branch-and-bound algorithm [7, 24] and mixed-integer programming (MIP) using IBM's OSL, Release 2 [13] were also applied to the same ISA problems. One difficulty in the comparison is that several equivalent MIP formulations exist which may obtain the same solution with different computational resources. The computational effort is also influenced in a rather unpredictable way by fixing the number of subsystems and their minimal size [13]. Another problem of the comparison is that in MIP the number of subsystems, the minimum number of columns per subsystem and the minimum number of rows per subsystem have been fixed whereas the GA selects suitable values for these parameters by itself. For a fair comparison, the best parameter values found by the GA were used to fix the accompanying parameters of the MIP models.

Under these preliminaries, the 7 cases regarded in Table 1 may be clustered in 3 groups. In the first 2 cases, the MIP-model and the BB were able to find solutions in rea-

Table 1. Results of tests

ISA problem[a]		Genetic algorithm (30 experiments per case)					MIP-OSL[d]			Specialized branch-and-bound[e]	
Dim of matrix	Number of base-blocks	Number of trials per experiment	Best fitness	Average fitness	Best fitness found in % of exper.	Average CPU-s[b]	Input-parameters (n,c,r)[c]	Objective function value	CPU-s	Objective function value	CPU-s
17×10	9	4000	6.80	6.80	100	1.00	3,3,3	6.80	180	6.80	0.08
12×07	10	10000	14.00	10.37	27	5.10	3,1,1	14.00	45	14.00	0.06
16×28	12	4000	12.00	11.80	37	1.60	6,2,1	11.60	f	12.00	0.22
19×12	16	6000	21.60	21.28	63	4.80	6,1,1	21.60	212400	21.60	0.28
28×18	24	15000	30.90	29.97	16	31.00	5,2,1	29.60	f	22.30	h
27×36	26	15000	36.40	33.29	3	32.00	7,1,1	27.40	f	15.10	h
37×24	28	25000	32.60	31.11	3	65.70	8,1,1	22.50	g	10.10	h

[a] Sources of problems solved: 17×10, 37×24: [18]; 12×7: [22]; 16×28: [3]; 19×12: [11]; 28×18: [20]; 27×36: [27]

[b] Average CPU-s for all trials of *one* experiment. To obtain the total CPU-time used for the GA, the numbers in this column have to be multiplied by the number of experiments, i.e., by 30

[c] Parameters determining the structure of the resulting matrix: n = number of subsystems, c = minimum number of columns per subsystem, r = minimum number of rows per subsystem

[d] Mixed-integer programming using IBM's OSL [13]

[e] Cf [24]

[f] Interrupted after several days (exceeding running time limit)

[g] Interrupted after several days (exceeding running time limit). [13] shows 6 different solutions to this problem for (n,c,r) = (5,3,3); the best objective function value obtained equals 26.2 (without proof of optimality). However, this value cannot be compared to the GA-solution due to different parameters (n,c,r)

[h] Interrupted due to memory overflow after an approximate CPU-time of 120 s

sonable time. However, the BB algorithm needed far less CPU-time than the OSL. For the first case, the total time taken for performing 30 experiments by the GA lay between the CPU-times of the BB and MIP whereas for case 2 the GA took the most time of all three procedures compared.

For the cases 3 and 4 which constitute group 2 the MIP-formulation run several days; only in one of the two cases it found the optimal solution within the given time limit. The BB again needed only fractions of one CPU-second to obtain the optimal solution! The GA took less than 3 minutes in both cases. With respect to the optimal objective function value, the GA found the optimal solution in every case which could be solved by exact optimization methods.

Group 3 consists of the cases 5 to 7 in which both exact optimization procedures failed either due to memory overflows (BB) or excessive running time (MIP). The optimal solutions of the problems in class 3 are still unknown. Both the BB and the MIP provided non-optimal solutions before stopping. However, the GA obtained in 2 of the 3 cases remarkably better solutions; in all three cases the GA needed far less computational time than the MIP. Taking the 27×36 matrix presented in [27], the BB reached the memory limit after 120 CPU seconds and was interrupted at a best objective function value of 15.10, the OSL was stopped after more than 4 days of calculation with a temporary best of 27.40 and the GA proposed a fitness value of 36.40 within 16 minutes (taken by all 30 experiments). Although the best fitness was only found in one experiment, the average value per experiment of 33.29 indicates that the GA is far better suited for complex problems than the exact methods.

The figures state also that the more complex the ISA problem, i.e., the more baseblocks and hence the larger the

length of the bit string, the lower the probability of finding the optimal fitness in one experiment. In cases 6 and 7 the GA found the best solutions only in 1 of 30 experiments although we chose a comparatively large number of trials per experiment. Each experiment with the GA leads the search into different solutions areas which may promote or restrain the evolution towards the global optimum.

In conclusion, the GA possesses a high potential for quickly converging towards good solutions for large ISA problems for which optimization methods fail. In our study the best fitness is usually not found in each GA experiment, but the average value over a series of 30 experiments is always fairly close to the best fitness and superior to the temporary best results delivered by exact methods.

5. Conclusion and outlook

This paper examines the application of a genetic algorithm to a clustering problem emerging from planning an information systems architecture. To solve this problem one has to rearrange the rows and columns of a BSP matrix with respect to several optimization criteria. Since exact methods fail to provide results for large ISA problems, the applicability of a GA is studied. This paper describes how a GA can be applied to the ISA problem and presents computational experiences with optimization techniques and a GA.

In conclusion, the results and the performance achieved with the GA are surprisingly good. The GA seems to be a genuine alternative to the application of exact methods for solving complex problems. It provides optimal solutions for those ISA problems which could be solved by exact methods and it determines feasible solutions for large problems which were not tractable by these procedures.

Furthermore, GA always provide solutions whereas the optimization methods may fail to deliver at least a non-optimal solution when stopped due to the lack of computational resources.

All automatically applied procedures are challenged by the speculation that a human decision maker may find better solutions, especially if he is supported by a decision support system. Therefore, an alternative to the procedures described for solving the ISA problem are online decisions on combining different baseblocks. Generally, only few results are available in which the performance of heuristics and of human decision-makers are compared in a scientifically sound setting. As in many other applications, experiments of such performance comparisons for determining IS architectures look promising but are fairly difficult to realize. Therefore, this comparison has not been considered in the present study.

References

1. Brathwaite KS (1992) Information Engineering, Vol. I. Concepts. CRC, Boca Raton
2. Brown DE, Huntley CL, Spillane AR (1989) A Parallel Genetic Heuristic for the Quadratic Assignment Problem. In: Schaffer JD (ed) Proceedings of the 3rd International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, pp 406–415
3. Flaatten PO et al. (1989) Foundations of Business Systems. Dryden, Chicago
4. Goldberg DE (1989) Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, Mass.
5. Grefenstette J (1991) Documentation for GENESIS. The Software Partnership
6. Hein KP (1985) Information System Model and Architecture Generator. IBM Syst J 24:213–235
7. Hess J (1993) Implementierung von Branch-and-Bound Verfahren zur Strukturierung von BSP-Matrizen. Master Thesis, University of Bern
8. IBM (ed) (1978) Business Systems Planning: Information Systems Planning Guide. IBM-Form, GE 20-0527-2
9. IBM (ed) (1990) Information System Model and Architecture Generator, Operation Guide, Release 1.3. IBM-Form SE 11-5989-2
10. Katz RL (1990) Business/enterprise modeling. IBM Syst J 29:509–525
11. Kiewiet DJ, Stegwee RA (1991) Conceptual Modeling and Cluster Analysis: Design Strategies for Information Architectures. In: DeGross JI et al. (eds) Proceedings of the 12th International Conference on Information Systems. ACM, Baltimore, pp 315–326
12. Kim YG, Everest GC (1994) Building an IS architecture. Inf & Manag 26:1–11
13. Knolmayer G (1994) The Application of Mixed Integer Programming to the "Business Systems Planning"-Problem. In: Dyckhoff H et al. (eds) Operations Research Proceedings 1993. Springer, Berlin, pp 457–463
14. Knolmayer G, Spahni D (1993) Darstellung und Vergleich ausgewählter Methoden zur Bestimmung von IS-Architekturen. In: Reichel H (ed) Informatik, Wirtschaft, Gesellschaft. Springer, Berlin, pp 99–104
15. Krcmar H (1990) Bedeutung und Ziele von Informationssystem-Architekturen. Wirtschaftsinformatik 32:395–402
16. Krovi R (1992) Genetic Algorithms for Clustering: A Preliminary Investigation. In: Nunamaker JF, Sprague RH (eds) Proceedings of the 25th International Conference on System Sciences, Vol. IV. IEEE, Los Alamitos, pp 540–544
17. Kusiak A, Chow WS (1987) An Efficient Cluster Identification Algorithm. IEEE Transactions on Systems, Man, and Cybernetics 17:696–699
18. Martin J (1982) Strategic Data-Planning Methodologies. Prentice-Hall, Englewood Cliffs
19. Niederman F, Brancheau JC, Wetherbe JC (1991) Information Systems Management Issues in the 1990s. MIS Quarterly 15:475–500
20. Orsey RR (1982) Methodologies for Determining Information Flow. In: Goldberg R, Lorin H (eds) The Economies of Information Processing, Vol. I. Wiley, New York, pp 57–70
21. Raz T, Yaung AT (1995) Application of clustering techniques to information systems design. Information and Software Technology 37:145–154
22. Schumann M, Schüte H, Schumann U (1994) Entwicklung von Anwendungssystemen. Springer, Berlin
23. Sowa JF, Zachman JA (1992) Extending and formalizing the framework for information systems architecture. IBM Syst J 31:590–616
24. Spahni D (1993) Solving the "Business System Planning"-Problem Using Specialized Branch & Bound Algorithms. In: Bachem A et al. (eds) Extended Abstracts of the 18th Symposium in Operations Research. Physica, Heidelberg, pp 491–494
25. Spahni D (1996) Verfahren zur Bestimmung geeigneter Teilsysteme integrierter Informationssysteme. Diss Universität Bern 1996
26. Spewak SH, Hill SC (1993) Enterprise Architecture Planning. QED, Wellesley
27. Teng JTC, Kettinger WJ, Guha S (1992) Business Redesign and Information Architectures: Establishing the Missing Links. In: DeGross JI, Becker JD, Elam JJ (eds) Proceedings of the 13th Intl. Conference on Information Systems. ACM, Baltimore, pp 81–89
28. Vetter M (1988) Strategie der Anwendungssoftware-Entwicklung – Planung, Prinzipien, Konzepte. Teubner, Stuttgart
29. Wetherbe JC, Davis GB (1983) Developing a long range information architecture. In: Smith AN, Medley DB (eds) Proceedings of AFIPS National Computer Conference, AFIPS, Arlington, pp 261–269